The VR Shopper Game Project - Gesture Based UI Development

Milosz Milewski, Ciaran Roche

B.Sc. (Hons) in Software Development

May 6, 2022

VR Game Project

Advised by: Prof. Damien Costello

Department of Computer Science and Applied Physics
Atlantic Technological University (ATU)

Ollscoil
Teicneolaíochta
an Atlantaigh

Atlantic
Technological
University

Contents

Chapter 1

About this project

Introduction *Project is available here: GitHub*

This is the development cycle documentation for our VR Unity Game project, the VR Shopper project - This document will outline the architecture structure, the learning process and rationale for our decisions taken whilst developing of this project.

Description The project is a farmer-themed VR game where the player is put in an endless-level - The user has two tools to their disposal, a hoe and a gardening can full of water - Along with that, they have two bags of seeds near their market stall - With these tools, the player has to figure out the planting and harvesting process, they can put their crops on the market stall, where randomly-spawned AI NPCs approach and take a random amount of vegetables from, and pay their due price.

Chapter 2

Project Implementation

Development Hardware To develop a VR application and to test it extensively, Milosz has used his personal Oculus Quest 2 for Unity development. Due to the constraint of one of the developers having such hardware, and the other one not having access to any VR hardware, some work had to be divided up based on that factor. Hence why Ciaran took up of more of an active role in developing visual assets as well as the systems that weren't tied to the Virtual Reality factor, such as generating the nav-mesh, visual set-up etc, as opposed to working on general programming of the VR-dependable systems.

During VR development, the game was streamed directly from Unity to the headset using Virtual Desktop, meaning that the actual machine running the project, was the development PC. At the time of writing it's unsure whenever the project correctly runs on the target headset when ran natively.

2.1 The Game

The Game Format The game consists of two main scenes - The main scene, as well as the gameplay scene - The game isn't divided into rounds or levels - the format was more so meant to be an ”endless” relaxing gameplay, trying to beat your last score.

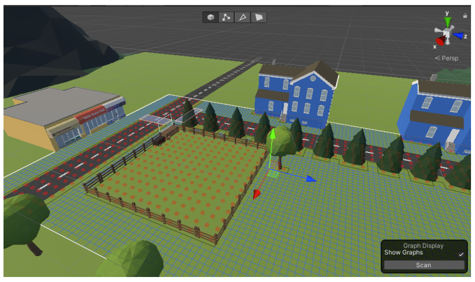

Figure 2.1: The Main Scene

Figure 2.2: The Menu scene

The Goal The challenge is self-imposed by the players, they could set-up hotseat rules where players swap out during rounds and try to achieve the highest score within a time-limit, or other.

Players plant and harvest various vegetables, then they display those vegetables on the market stall table that is provided to them - AI NPCs then walk by the table and purchase a random amount of vegetables from the stall, giving the player points.

There's a form of plant management - where you have to make sure the plants have enough water so they do not die, players also need to make sure that they do not plant a lot of vegetables at once, since that can easily overwhelm less experienced players.

2.1.1 Gestures

There are several gestures used in the application. Using the controllers with the VR set made it very easy to identify potential gestures as it would be almost exactly as if it were real life.

Because of the application being developed for the VR hardware – Users can use virtually any hand gesture they wish to do so – The player movement is not inhibited in any way, and the user can choose their gestures freely.

However, there are a few functionally distinct "trigger" gestures the application listens for to provide functionality to certain actions the player takes, which will be outlined below.

1. The hoe digging gesture - The player must hit the ground, with the top (head) part of the hoe to dig a pile up. There are certain conditions that must pass for the action to be performed successfully.
   (a) The velocity at the point of impact must be greater than 15 Unity units, meaning small arm movements won't trigger the mud pile

   (b) The hoe tool must be currently grabbed by the user, simply dropping the hoe on the ground with high enough velocity won't perform the action.

   (c) The hit contact point must be within 3 Unity distance units of the tool head, hitting the ground with the handle won't do

   (d) The hit game object must be of layer "TerrainDiggable", you won't be able to dig on any other game object

   (e) The hit contact point must pass a collider check – You won't be able to create a new mud pile if there are any entities of layer "Entities" within that sphere collider check – This ensures the mud-piles cannot be overlaid on top of each other, as well as ensures proper clearance around the mud piles

2. Teleporting - The player has a RayInteractor component attached to his right controller, meaning, a visual ray is shooting out of the controller. The player must reach out and on the ground in the direction he wishes to move to, then the player should press the primary button (A on oculus) to teleport. The teleport location will be the hit point of that ray, but only if it's hitting a valid Teleportation mesh (There's only one in the whole game, limiting the play area)

3. Planting the seeds - The player is completely unrestricted in the way they wish to plant the seed. They could pick up a seed, drop it, throw it, put it on the mud pile, and it'll be considered planted – Hell, you can even put the seed on the ground, and use the hoe as your golf stick to punt the seed into the mud pile. That's the freedom of VR.

4. Picking up items - Reach out, bend down, and pick up the item using the grip button - You can then juggle it, put it down, throw it - whatever the user desires. Some items have dedicated hold-points that the hand snaps to improve user experience. For example, when picking up the gardening can, the gardening can reposition itself so that the users' hand is on the handle

5. Gardening can usage - You can pick up the gardening can as usual, except its usual interaction (water emitting-particles) only occur when the gardening can is being actively held AND is tilted over.

6. Interacting with UI elements - Very similar to teleportation, reach out towards the UI button in the world, and let the ray hit the UI element, then pull trigger to interact with the element.

As said before, VR offers infinitely more ways to interact with your game environment than traditional input devices such as keyboard and mouse - However, this also means, that in order to make such interactions look realistic, you have to overcome a new set of problems - You have to implement constraints to make those actions look believable - For example, as seen with the hoe, I have had to make sure that the user used the proper part of the hoe to dig the terrain, and that they did it with enough force, rather than tapping the ground lightly.

Same goes for everything else - Players will often seek ways to break the game, and even if you assume that ”surely, nobody would grab the glass upside down and try to drink from it”, then you should expect the player to grab the glass upside down, and try to drink from it that way. Such interaction wouldn't needed to be accounted for usually, because in traditional games interactions are driven, usually, by animations performed by your character, rather than your actual 1:1 body movement.

In short, when designing for VR, you to remember that the VR user *is the player*, where in traditional games the user merely *controls the player* in a way that's modelled by the developer.

2.2 Architecture of the Solution

Here we will explain some of the most important libraries and tools that have driven the development further.

2.2.1 Dependencies, Libraries and Unity Packages

Open XR OpenXR is a very useful layer between all the different VR runtimes you could target your application for. When targeting a VR application for OpenXR, you can automatically build your games for various VR runtimes, such as OpenXR, Oculus, WMR (Windows Mixed Reality) etc.

It allows you to write code once, and being sure that your development is not platform-dependent and runs as expected on all major VR headsets.

The very brains and heart of the VR functionality in this project.

XR Interaction Toolkit This Unity package is almost a must-have for working with OpenXR - It does the ground-work of inter-entity interactivity and provides a high-level functionality implementation for handling things such as item-pick up, controller set-up, item hover/activation and so on.

If Open XR is the heart and brains of the VR functionality, then this should be considered as the skeleton and "limbs" of the architecture, enabling the VR interactivity.

Worth nothing that this package is recommended by Unity itself for interactions - developing those features yourself is a very low-level task and is generally not recommended to do it yourself, if all you're seeking to do is to create a simple VR game. Even the official Unity courses use the XR Interaction Toolkit for basic interaction.

A* Pathfinding Project An Unity package implementing very performant and feature-rich implementation of A* (not only) pathfinding algorithm. It has let us set-up nav-meshes for our AIs to navigate on. Our state-machine AIs use exposed to us by this library Seeker components to navigate on the nav-mesh.

2.3 Code Structure and Architecture

A simplified chart of interoperability between systems This chart is not meant to be fully accurate - Classes will be missing, methods are not shown - It's supposed to show a general idea of how systems interact with each other, as well as the general class responsibility.

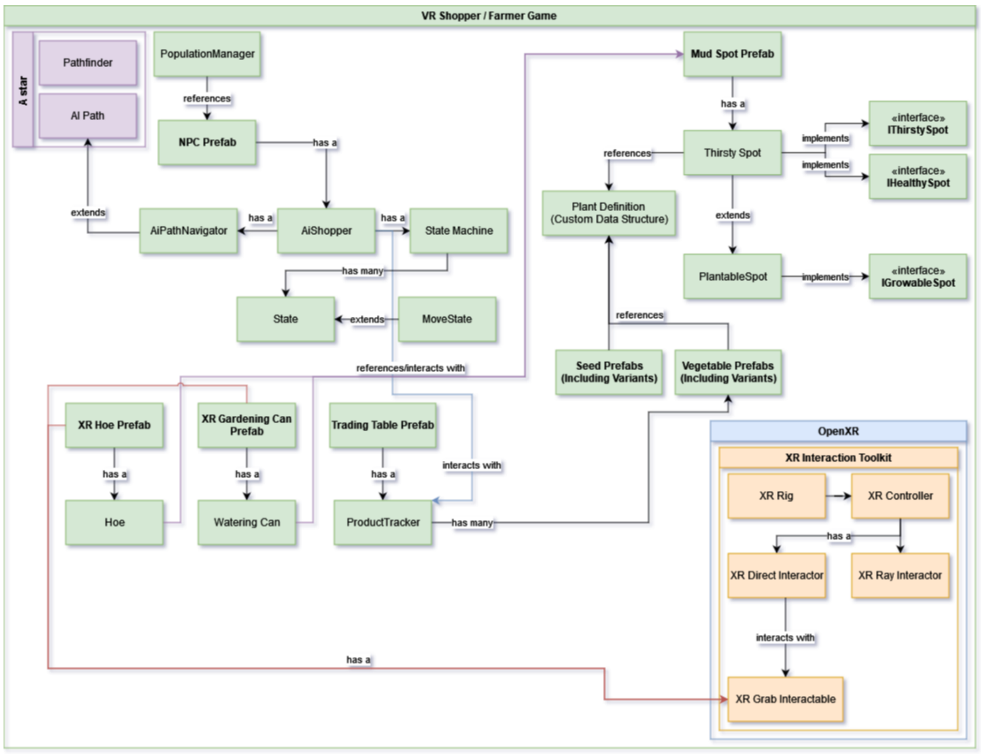

Figure 2.3: The system Diagram

Honestly, generating UML diagrams for Unity isn't the best solution as Unity heavily uses the Entity Component pattern, those relationships are often hard to show on a diagram, and when they are shown, they usually provide far too much information, ending up cluttering it.

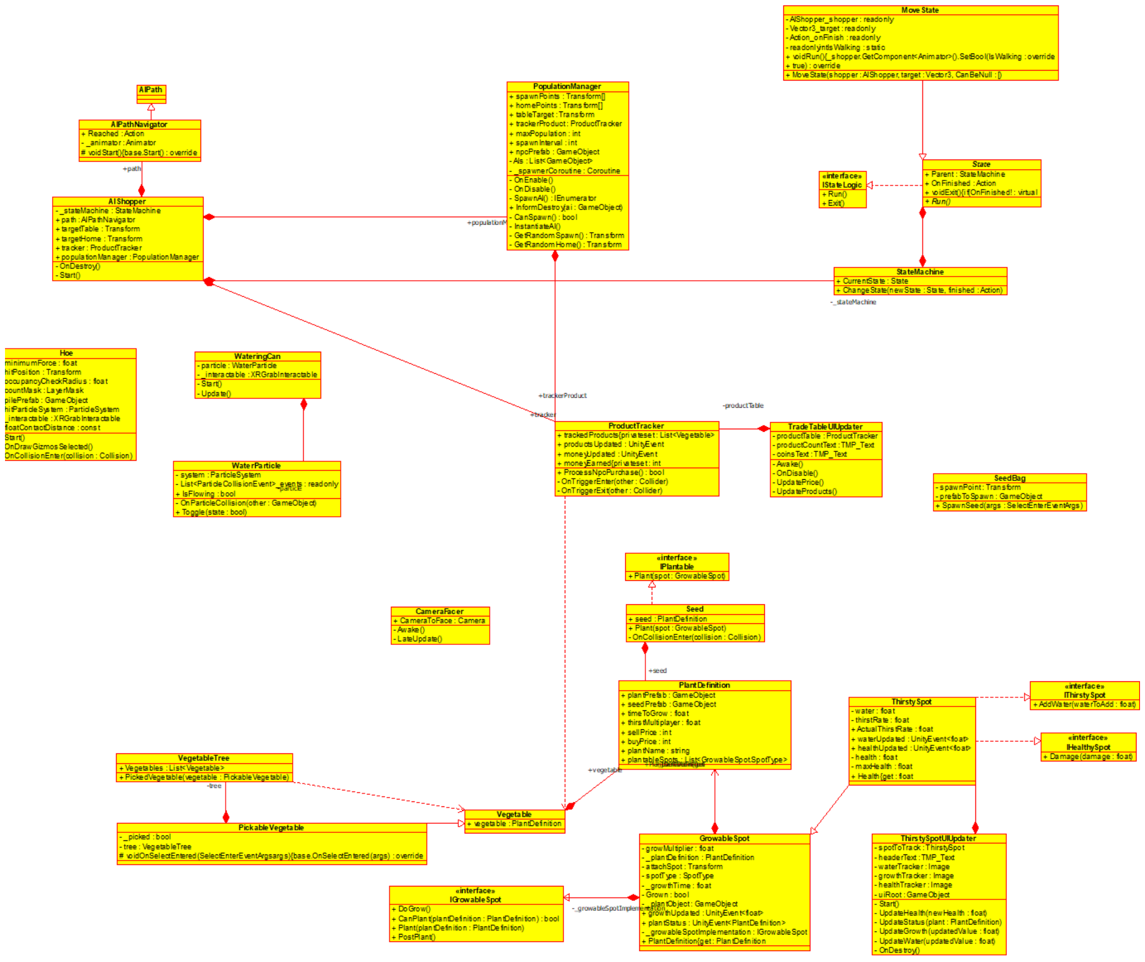
Move State
- AIShopper_shopper : readonly
- Vector3_target : readonly
- Action_onFinish : readonly
- readonlyintIsWalking : static
+ voidRun(){_shopper.GetComponent<Animator>().SetBool(IsWalking : override
+ true) : override
+ MoveState(shopper : AIShopper, target : Vector3, CanBeNull : [])
AIPath
AIPathNavigator
+ Reached : Action
- _animator : Animator
# void Start(){base.Start() : override
+path
PopulationManager
+ spawnPoints : Transform[]
+ homePoints : Transform[]
+ tableTarget : Transform
+ trackerProduct : ProductTracker
+ maxPopulation : int
+ spawnInterval : int
+ npcPrefab : GameObject
- AIs : List<GameObject>
- _spawnerCoroutine : Coroutine
- OnEnable()
- OnDisable()
- SpawnAI() : IEnumerator
+ InformDestroy(ai : GameObject)
- CanSpawn() : bool
- InstantiateAI()
- GetRandomSpawn() : Transform
- GetRandomHome() : Transform
«interface»
IStateLogic
+ Run()
+ Exit()
State
+ Parent : StateMachine
+ OnFinished : Action
+ voidExit(){if(OnFinished! : virtual
+ Run()
AIShopper
- _stateMachine : StateMachine
+ path : AIPathNavigator
+ targetTable : Transform
+ targetHome : Transform
+ tracker : ProductTracker
+ populationManager : PopulationManager
- OnDestroy()
- Start()
+populationM
StateMachine
+ CurrentState : State
+ ChangeState(newState : State, finished : Action)
-_stateMachine
Hoe
minimumForce : float
hitPosition : Transform
occupancyCheckRadius : float
countMask : LayerMask
pilePrefab : GameObject
hitParticleSystem : ParticleSystem
_interactable : XRGrabInteractable
floatContactDistance : const
Start()
OnDrawGizmosSelected()
OnCollisionEnter(collision : Collision)
WateringCan
- particle : WaterParticle
- _interactable : XRGrabInteractable
- Start()
- Update()
+trackerProduct
+tracker
-productTable
ProductTracker
+ trackedProducts{privateset : List<Vegetable>
+ productsUpdated : UnityEvent
+ moneyUpdated : UnityEvent
+ moneyEarned{privateset : int
+ ProcessNpcPurchase() : bool
- OnTriggerEnter(other : Collider)
- OnTriggerExit(other : Collider)
TradeTableUIUpdater
- productTable : ProductTracker
- productCountText : TMP_Text
- coinsText : TMP_Text
- Awake()
- OnDisable()
- UpdatePrice()
- UpdateProducts()
WaterParticle
- system : ParticleSystem
- List<ParticleCollisionEvent>_events : readonly
-particle
+ IsFlowing : bool
- OnParticleCollision(other : GameObject)
+ Toggle(state : bool)
SeedBag
- spawnPoint : Transform
- prefabToSpawn : GameObject
+ SpawnSeed(args : SelectEnterEventArgs)
«interface»
IPlantable
+ Plant(spot : GrowableSpot)
CameraFacer
+ CameraToFace : Camera
- Awake()
- LateUpdate()
Seed
+ seed : PlantDefinition
+ Plant(spot : GrowableSpot)
- OnCollisionEnter(collision : Collision)
+seed
PlantDefinition
+ plantPrefab : GameObject
+ seedPrefab : GameObject
+ timeToGrow : float
+ thirstMultiplayer : float
+ sellPrice : int
+ buyPrice : int
+ plantName : string
+ plantableSpots : List<GrowableSpotSpotType>
«interface»
IThirstySpot
+ AddWater(waterToAdd : float)
ThirstySpot
- water : float
- thirstRate : float
+ ActualThirstRate : float
+ waterUpdated : UnityEvent<float>
+ healthUpdated : UnityEvent<float>
- health : float
- maxHealth : float
+ Health{get : float
«interface»
IHealthySpot
+ Damage(damage : float)
VegetableTree
+ Vegetables : List<Vegetable>
+ PickedVegetable(vegetable : PickableVegetable)
-tree
+vegetable
Vegetable
+ vegetable : PlantDefinition
PickableVegetable
- _picked : bool
- tree : VegetableTree
# voidOnSelectEntered(SelectEnterEventArgsargs){base.OnSelectEntered(args) : override
GrowableSpot
- growMultiplier : float
- _plantDefinition : PlantDefinition
- attachSpot : Transform
- spotType : SpotType
- _growthTime : float
- Grown : bool
- _plantObject : GameObject
+ growthUpdated : UnityEvent<float>
+ plantStatus : UnityEvent<PlantDefinition>
- _growableSpotImplementation : IGrowableSpot
+ PlantDefinition{get : PlantDefinition
ThirstySpotUIUpdater
- spotToTrack : ThirstySpot
- headerText : TMP_Text
- waterTracker : Image
- growthTracker : Image
- healthTracker : Image
- uiRoot : GameObject
- Start()
- UpdateHealth(newHealth : float)
- UpdateStatus(plant : PlantDefinition)
- UpdateGrowth(updatedValue : float)
- UpdateWater(updatedValue : float)
- OnDestroy()
«interface»
IGrowableSpot
+ DoGrow()
+ CanPlant(plantDefinition : PlantDefinition) : bool
+ Plant(plantDefinition : PlantDefinition)
+ PostPlant()
-_growableSpotImplementation

Figure 2.4: The UML Diagram

Chapter 3

Conclusion

Retrospection The team worked extremely well and good, daily communication was maintained throughout the development process of it - The team split the responsibilities based on constraints and the skills of the team member. Any and all issues that we encountered during development were resolved quickly and swiftly.

Big decisions were made at the start of the development cycle *(We actually changed the theme of the project early because of lack of assets)*, and were well discussed beforehand. In the end, both Ciaran and Milosz were happy with the completeness and state of the project.

VR Development Overall, development of feature systems is largely the same as usual systems, except the ones that the user is directly, physically interacting with - Suddenly, you have to enforce how the user is going to interact with your entity, define a set of rules that pass that interaction - You will never know how the user will interact with your system, and there are hundreds more possible ways the user can express themselves in VR.

This is an excellent thing, giving you many, many more ways in letting the user "consume" such an interaction with your game-world - Not to mention, that even the most boring games created for VR gain that novelty status from being in VR in itself.

Exception to that rule is when you don't have any VR hardware - additional meet-up sessions had to be created in order to let the other user experience and play-test the game.

3.1 Learning Outcomes

3.1.1 Milosz

I have been interested in developing for VR in a long while - I've finally had the reason to put a little bit of my time into it and it's been a very good experience. Development for VR systems is made easy by tools such as OpenXR and XR Interaction Toolkit and it's been a breeze developing most of the systems for this game. I enjoyed working with Unity and the Oculus Quest 2.

This is also the first time I have worked with path-finding in Unity - Ciaran has set-up the navmesh, whilst I learned how to set-up the AI agents to navigate on it - Weirdly enough, it was an easy process due to the library we've used. I have even applied my previously gained knowledge of state-machines to develop a rather simple AI in our game.

Collaborating with Ciaran, and not having to worry about the visual aspects of the project, lifted a huge weight off my shoulders and allowed me to focus on the main gameplay systems.

3.1.2 Ciaran

I've learned a lot about the Oculus Quest hardware and the possibilities it unlocks and the degrees of freedom it gives users playing games. It was a completely new experience for me and has intrigued me to learn more about the VR world.

It's been a shame I couldn't VR play-test more, but at least I have learned how I could contribute to a VR project in other ways that programming VR systems - I've learned a lot about importing animations into Unity and setting them up with state machines, process that was a lot simpler than I've previously thought.

Bibliography